НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА ЭКОНОМИКИ

Домашнее задание #1

по дисциплине
«Основы iOS разработки»

Москва 2024

Тема: Создание первого iOS приложения.
Цель: Изучить работу со Storyboard и получить опыт работы с Xcode и Swift.

Описание задания

Чтобы выполнить данное домашнее задание, вы должны реализовать iOS приложение, используя один storyboard. Приложение содержит один экран с множеством view и одной кнопкой. Нажатие на кнопку ведет к изменению цвета и формы всех view.

Требования

- Задание должно быть выполнено с использованием Storyboard и Swift.
- Требования к архитектуре не предъявляются для этого задания.
- Прежде чем использовать код из этого туториала или интернета, вы должны сначала его понять.

Оценивание

Оценка	Задание
1	Приложение содержит storyboard, кнопку и все view.
2-3	Нажатие на кнопку изменяет цвета view.
4	Цвет каждого view уникален.
5	Нажатие на кнопку приводит как к изменению цвета, так и к изменению corner radius.
6	Изменение цвета происходит анимировано.
7	Кнопка неактивна в момент анимации изменения цвета.
8	View расставлены самостоятельно интересным и креативным способом.
9	Ваш код не содержит магических чисел и содержит метки.
10	Случайные цвета сгенерированы в HEX, а затем сконвертированы в RGB с использованием расширения UIColor.

Дисклеймер

Просрочка сдачи работ **наказывается** понижением оценки на количество баллов, равное количеству дней с момента дедлайна. Штраф ограничивается **минус 5 баллами**. Пожалуйста, помните об этом и планируйте вашу неделю соответственно.

Не стесняйтесь задавать любые вопросы, касающиеся этой задачи или iOS разработки в целом. Наша цель – помочь вам стать **лучшими iOS разработчиками**, насколько это возможно. Преподаватели и ассистенты здесь чтобы **помочь!**

Работы, содержащие необычные решения, исключительный стиль кода, использование шаблонов и интересные функции, могут претендовать на бонусные баллы. Это не подлежит обсуждению. Повторение и копирование решений, за которые ранее были заработаны бонусные баллы, не дает бонусных баллов.

Начиная с домашнего задания #3, нарушение кодстайла и некачественные решения могут привести к снижению оценки.

Студенты, которые уже умеют анимировать и создавать UI могут обратиться к преподавателю за **более сложным** домашним заданием.

Туториал

Шаг 1:

Откройте Xcode и создайте новый проект (Рисунок 1)

Рисунок 1

В следующем всплывающем окне выберите платформу iOS и опцию App в разделе «Application» (Рисунок 2)

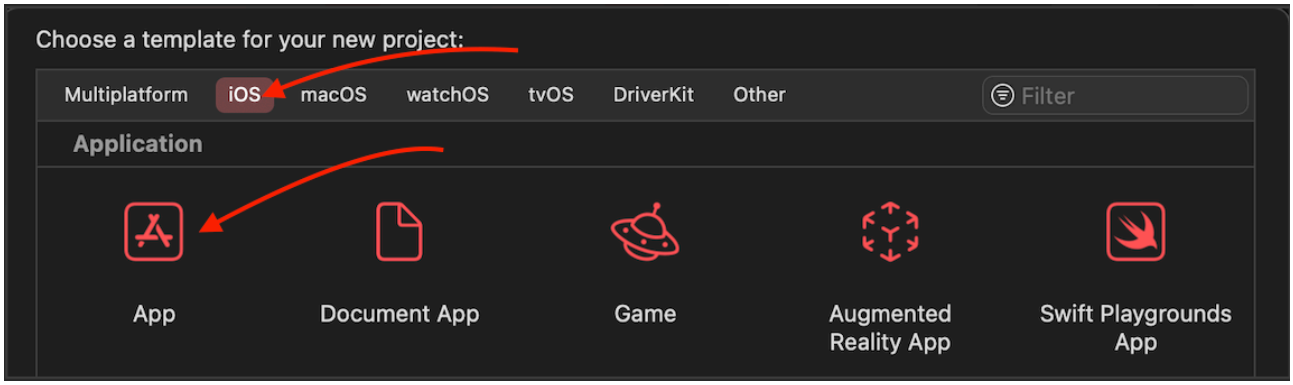

Рисунок 2

Назовите свой проект следующим образом: *адрес вашей учебной электронной почты до «@»*PW*номер домашнего задания*. Пример: gmsosnovskiyPW1. **Убедитесь,** что вы выбрали **Storyboard** в качестве способа реализации интерфейса для этого задания. Нажмите Далее и создайте приложение.

Выберите файл Main в левом боковом файловом меню. Это наш storyboard. Далее, найдите и нажмите на иконку со знаком плюс в правом верхнем углу. В появившемся окне выберите кнопку, которая вам нравится (Рисунок 3), возьмите и перетащите ее на экран iPhone. Можете установить произвольное название и расположение кнопки на экране.

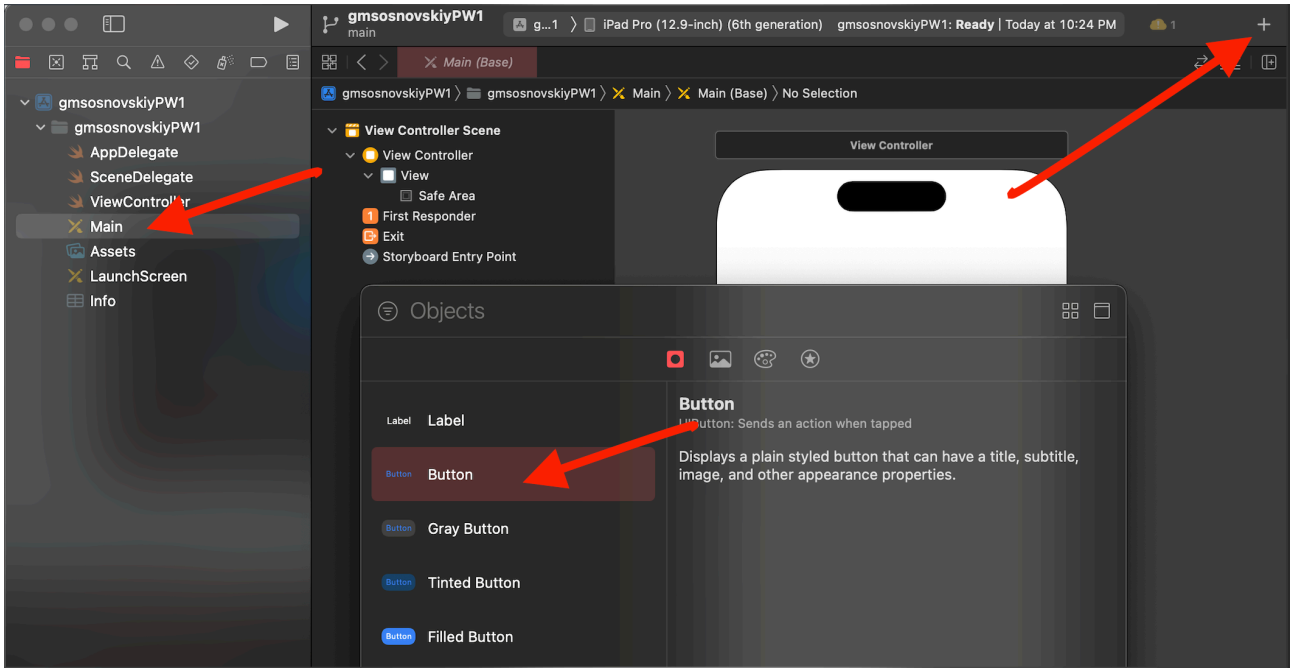

Рисунок 3

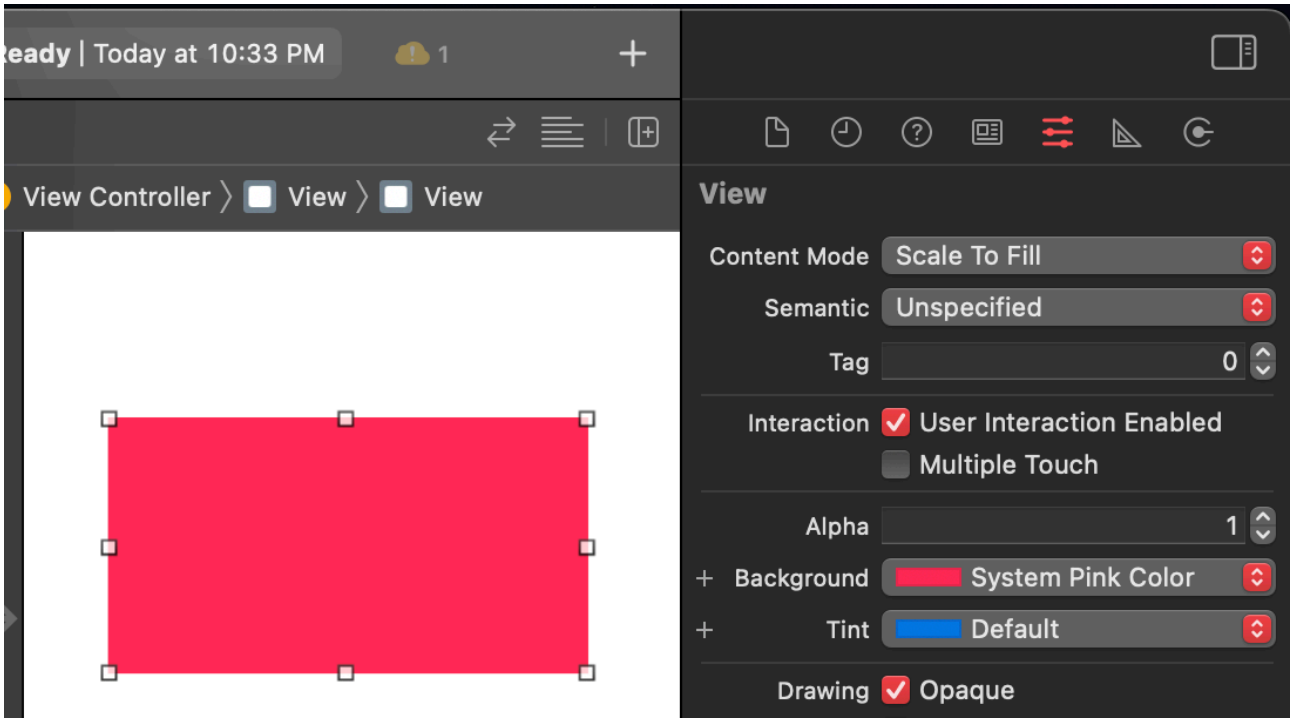

Рисунок 4

В том же самом всплывающем окне найдите view и перетащите несколько (минимум 3, но не стесняйтесь выходить за границы возможного!) на View Controller. Выберите цвет фона ваших view для того, чтобы они выделялись на экране (Рисунок 4).

Шаги 2-3:

Мы должны связать все кнопки и все view на нашем view controller-е с определенным местом в коде. Чтобы это сделать, зажмите клавишу option (alt) и откройте файл ViewController (над файлом Main на Рисунке 3). Это откроет его в параллельном режиме со storyboard-ом. Теперь задержите клавишу control (не перепутайте клавиши command и control) и перетащите все view прямо под определение класса ViewController (Рисунок 5).

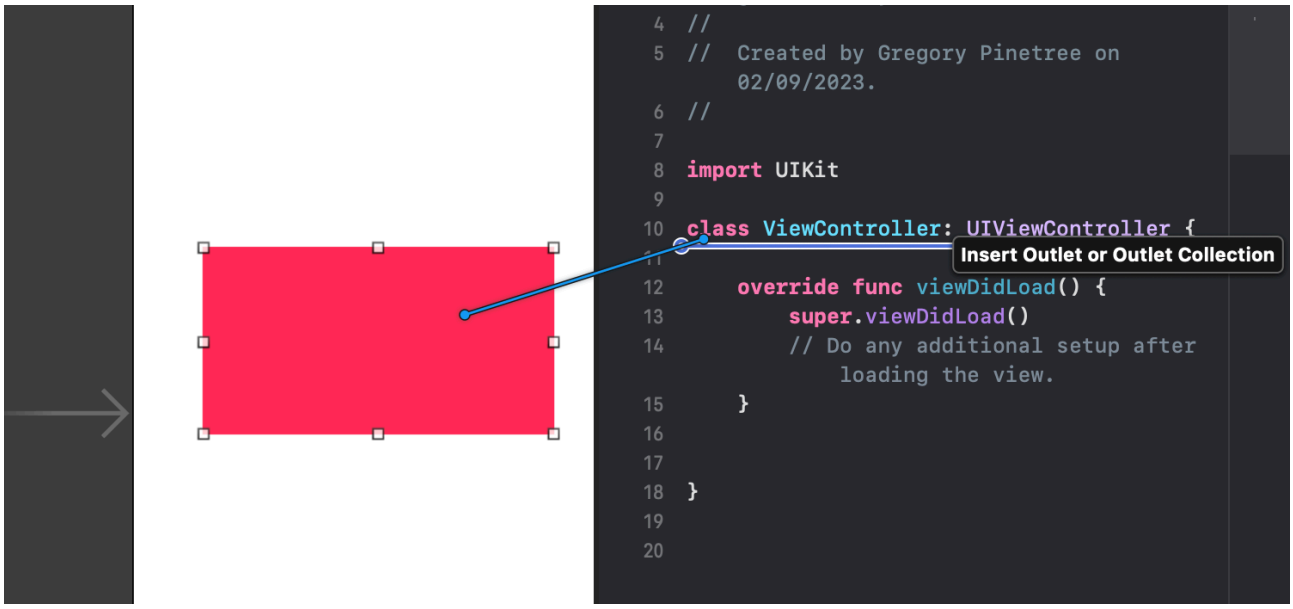

Рисунок 5

Вы можете назвать их отдельно или создать массив view. В первом случае вы должны быть аккуратны, чтобы не перетащить view в уже существующий outlet, а создать индивидуальный outlet для каждого view. Во втором случае вам нужно перенести все view в один единственный массив.

Способ 1:

```
    @IBOutlet weak var view1: UIView!
    @IBOutlet weak var view2: UIView!
    @IBOutlet weak var view3: UIView!
```

Способ 2:

```
    @IBOutlet var views: [UIView]!
```

Это позволяет нам обращаться к view как к полям в классе ViewController. Нам пока не нужно поле в классе для кнопки, но нам необходимо предпринимать какие-то действия при нажатии пользователя на нее. Таким образом мы, удерживая клавишу control, перетаскиваем нашу кнопку на одну строку ниже окончания функции **viewDidLoad**. Убедитесь, что связь является действием (IBAction). Назовите эту функцию **buttonWasPressed**, так как она будет вызываться при нажатии пользователя на кнопку.

С помощью этого метода мы хотим изменять цвета наших view. Это можно осуществить, установив случайный UIColor в свойство backgroundColor у view. К сожалению, у UIColor нет метода .random() (Может кто-то из трудолюбивых студентов может это изменить?), поэтому нам нужно написать свой собственный.

Те, кто использует массив для хранения view, должны самостоятельно разобраться с тем как использовать “**for”**.

```
view1.backgroundColor = UIColor(
    displayP3Red: .random(in: 0...1),
    green: .random(in: 0...1),
    blue: .random(in: 0...1),
    alpha: 1
)
```

Это изменит цвет view по нажатию кнопки. Но впереди еще долгий путь.
Вопрос: Почему это не будет гарантировать уникальные цвета у view?

Шаг 4:

Существует множество способов убедиться, что цвета уникальны. Лучшие студенты создадут отдельный метод getUniqueColors(), который возвращает массив случайных уникальных цветов, но в этом туториале мы выберем самое простое возможное решение:

```
        var set = Set<UIColor>()
        while set.count < views.count {
            set.insert(
                UIColor(
                    red: .random(in: 0...1),
                    green: .random(in: 0...1),
                    blue: .random(in: 0...1),
                    alpha: 1
                )
            )
        }

        for view in views {
            view.backgroundColor = set.popFirst()
        }
```

На этот раз, людям с отдельными полями для view придется самостоятельно разобраться с тем, как подстроить код под себя. Вот такой сюжетный поворот.

Пожалуйста, не просто копируйте этот код в свой проект! Попытайтесь придумать свой подход. Вы сильные и независимые студенты, и вам не нужно это решение, чтобы достичь величия, верно?

Шаг 5:

Чтобы изменить corner radius у view, вы можете сделать следующее:

```
    view.layer.cornerRadius = .random(in: 0…25)
```

То, как вы сделаете это со всеми своими view, остается за вами. Некоторые могут подумать, что будет лучше выделить такую логику в новую функцию (И они правы!).

Шаг 6:

Animation – это метод, который принадлежит классу UIView. Важно знать, что любое изменение в UI приложения должно происходить в главном потоке (main thread) (Потоки будут изучены позже).

У UIView есть метод UIView.animate(withDuration: TimeInterval, animations: {}), это позволит нам анимировать изменения UI.

Первый параметр – время анимации в секундах (тип TimeInterval – это typealias для double. Кстати, что такое typealias?).

Второй параметр – это сама анимация. Это достигается с помощью передачи анонимной функции (Вы уже изучали анонимные функции, верно? Если нет, попытайтесь найти информацию о них самостоятельно или обратитесь к нам за помощью!).

Наша анимация изменения цвета будет выглядеть примерно вот так:

```
UIView.animate(withDuration: 3.49, animations: {
    view1.backgroundColor = set.popFirst()
    view1.layer.cornerRadius = .random(in: 0...25)
})
```

Шаг 7:

Если мы глубже исследуем функцию .animate, мы сможем найти спрятанный параметр в его семантике, так называемый completion. Это опциональная анонимная функция (анонимные функции в Swift называются callbacks), которая будет выполняться как только завершится анимация. Итак, все что нам нужно сделать – это деактивировать кнопку вначале метода buttonWasPressed и активировать ее в completion:

```
UIView.animate(
    withDuration: 0.21,
    animations: {
        view1.backgroundColor = set.popFirst()
        view2.layer.cornerRadius = .random(in: 0...25)
    },
    completion: { [weak self] _ in
        self?.button.isEnabled = true
    }
)
```

Вот, где вы можете почувствовать себя потерянными. Вы не знаете, что такое «[weak self]», почему здесь используется «_ in» и что это означает? «self?.» Вызывает экзистенциальный кризис. iOS разработка – это страшно, но, пожалуйста, не прячьтесь под стол. Все это будет рассмотрено в курсе позже. Все, что вам нужно знать – эти вещи касаются управления памятью в Swift.

Есть два способа обратиться к кнопке. Первый – сделать то же самое, что мы делали с view и создать outlet для кнопки, сделав его полем в ViewController. Или вы можете сделать это:

```
let button = sender as? UIButton
```

Если вы воспользуетесь вторым вариантом, у вас в коде не будет «[weak self]» или «self?.», но вы должны будете подвергать свою кнопку сомнению каждый раз при использовании.
Пример: button**?**.isEnabled = false.

Заключение:

При запуске наши view будут простого цвета и без закругленных углов. Чтобы избежать этого, мы можем вызвать методы для рандомизации цветов и изменения радиуса в методе viewDidLoad после super.viewDidLoad().

Если вы успешно завершили этот туториал, то вы можете получить за это домашнее задание **до 7 баллов**. Это хорошая оценка, но чтобы заработать больше, вам нужно выполнить оставшиеся шаги самостоятельно. Вам нужно будет изучить такие вещи, как auto-resizing и auto-layout. Вам нужно будет поработать с расширениями (extensions), проявить немного творчества, и продемонстрировать свои навыки программирования. **Удивите** того, кто будет проверять ваши домашние задания, получите бонусные баллы и уважение!

Многие шаги этого туториала носят чисто образовательный характер и их можно пропустить, но это будет невозможно без предварительного прочтения задания. В следующий раз, возможно, вам захочется **прочитать задание, прежде чем его выполнять.**

Пример законченного приложения: